JavaPOS Interface for FPE Terminals

Interface Guide

symbol®

The Enterprise Mobility Company ™

JavaPOS Interface for FPE Terminals

Interface Guide

72E-91668-01

Revision A
November 2006

Symbol Technologies, Inc.
One Symbol Plaza
Holtsville, New York 11742-1300
http://www.symbol.com

Revision History

Changes to the original manual are listed below.

Change	Date	Description
-01 Rev A	11/2006	Initial Symbol Release (Hypercom Version 1.1)

Table of Contents

About This Guide

Introduction

This document describes JavaPOS interface implementation details for Symbol PD87xx/PD47xx PIN pads (such as PD8700, PD4700 and PD4750) using FPE as the communication protocol between terminal and host. The supported POS objects are:

PIN pad
MSR (Magnetic stripe reader)
Signature Capture
Keyboard
Line Display

The implementation fits Unified POS version 1.7 standards described in http://www.nrf-arts.org/.
POS object implementations are written with using Sun JDK Version 1.3.

✓ ***NOTE:*** Screens and windows pictured in this guide are samples and can differ from actual screens.

! ***IMPORTANT:*** Any references in this guide to Hypercom Corporation, Hypercom logo, Hypercom file names and file paths, Hypercom software and terminals reflect hardware and software manufactured by Hypercom Corporation for Symbol Technologies, Inc.

Notational Conventions

The following conventions are used in this document:

If applicable, the term “FormBuilder” in this guide refers to software.
Italics are used to highlight the following:
- o Chapters and sections in this and related documents
- o Drop-down list and list box names
- o Check box and radio button names
- o Icons on a screen.

Bold text is used to highlight the following:
- o Names of windows
- o Dialog box components.

bullets (•) indicate:
- o Action items
- o Lists of alternatives
- o Lists of required steps that are not necessarily sequential

Sequential lists (e.g., those that describe step-by-step procedures) appear as numbered lists.
Special icons:

✓ ***NOTE:*** Notes contain neutral or positive information supplementing the main text. It is often information that applies only to special cases.

! ***IMPORTANT:*** Important statements draw attention to information crucial to using the product successfully. Pay special attention to Important statements.

! ***CAUTION***: Cautions advise that a negative result, such as a loss of data, may occur.

WARNING: Warnings provide information that is essential to the safety of the user, the equipment, or both. Failure to do as instructed may result in physical damage.

Related Documents

For the latest version of this and all payment solutions guides, go to: http://www.symbol.com/manuals .

Service Information

For service information, warranty information, technical assistance or problems with the equipment, contact the regional Symbol Global Customer Interaction Center in your area by visiting: www.symbol.com/contactsupport . Before calling, have the model number, serial number and several bar code symbols at hand.

Call the Global Customer Interaction Center from a phone near the scanning equipment so that the service person can try to troubleshoot the problem. If the equipment is found to be working properly and the problem is reading bar codes, the Support Center will request samples of the bar codes for analysis at our plant.
If the problem cannot be solved over the phone, it may be necessary to return the equipment for servicing. If that is necessary, the Global Customer Interaction Center will provide specific directions.

✓ ***NOTE:*** Symbol Technologies is not responsible for any damages incurred during shipment if the approved shipping container is not used. Shipping the units improperly can possibly void the warranty. If the original shipping container was not kept, contact Symbol to have another sent.

If the Symbol product was purchased from a Symbol Business Partner, contact that Business Partner for service.

Component Model

Figure 1 and **Figure 2** show the base components included in the system.

Terminal – Any Symbol PD8700 terminal with running FPE application

IODriver – Application running in a separate process and working as Input/Output bridge between Terminal and FPEInterface controllers. However it can run in the same process as User Application in some cases. The IODriver have two base implementations differing in communication type. One is Serial port driver and another one is TCP/IP port driver. The IODriver allows sharing one communication port among multiple user applications.

FPEInterface – Service object providing functional interface to the user components. The user components can call methods of FPEInterface and it translates the methods and parameters into the formatted FPE protocol data packets and submits to IODriver. It also works in the opposite direction getting output from IODriver and converting it to understandable event objects that are fired further to the user components.

POS devices – MSR, PINPad, LineDisplay, Keyboard, Signature Capture devices working according to Unified POS standard and use FPEInterface functions for implementing their functionality.

User Application – Any user application that uses JavaPOS objects. The User Application 2 on the diagram uses also FPEInterface and IODriver directly, that is not Unified POS standard, but sometimes it can help the user application to access more sophisticated terminal functions.

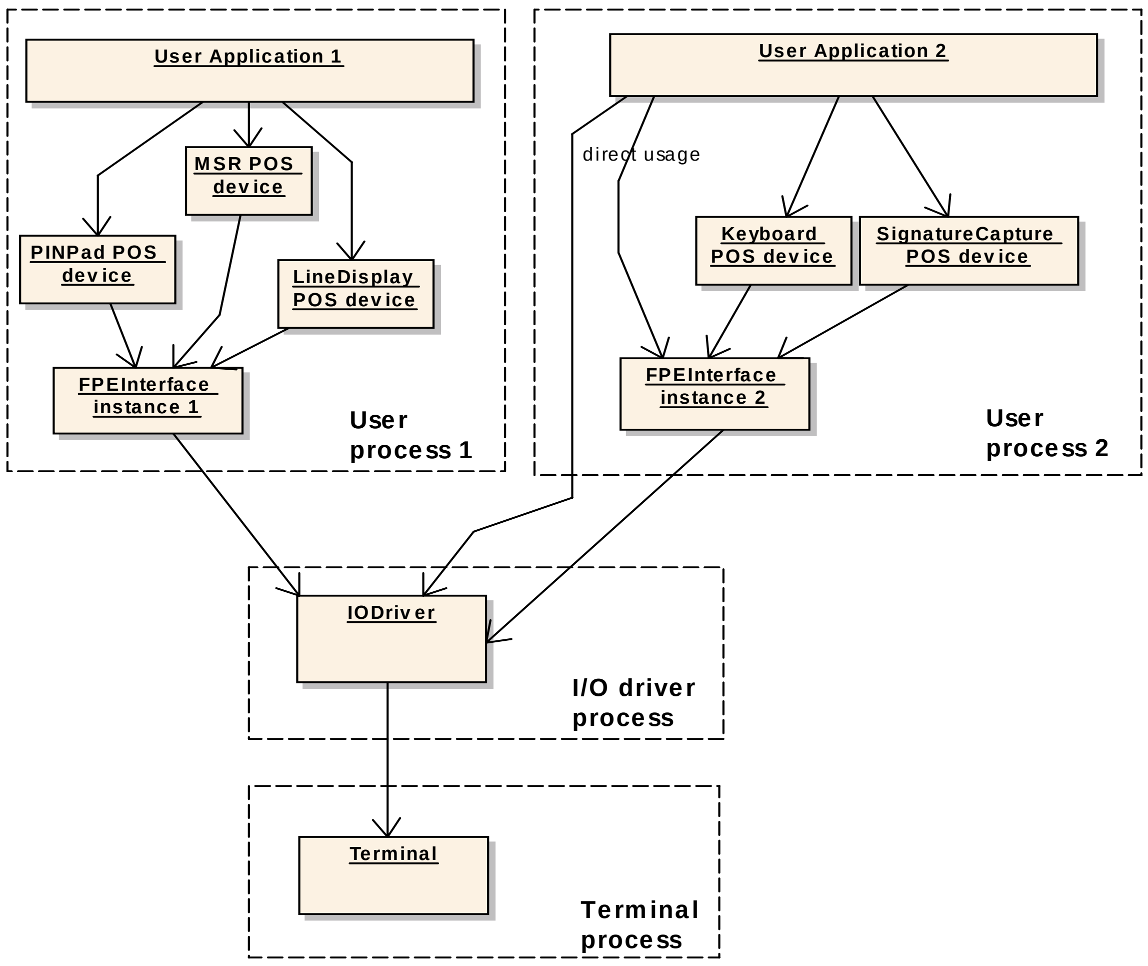


Figure 1 *Symbol JavaPOS Iinternal Component Model. This diagram shows an example of how JavaPOS objects can be used by two applications running simultaneously when terminal is connected via serial (RS232) port.*

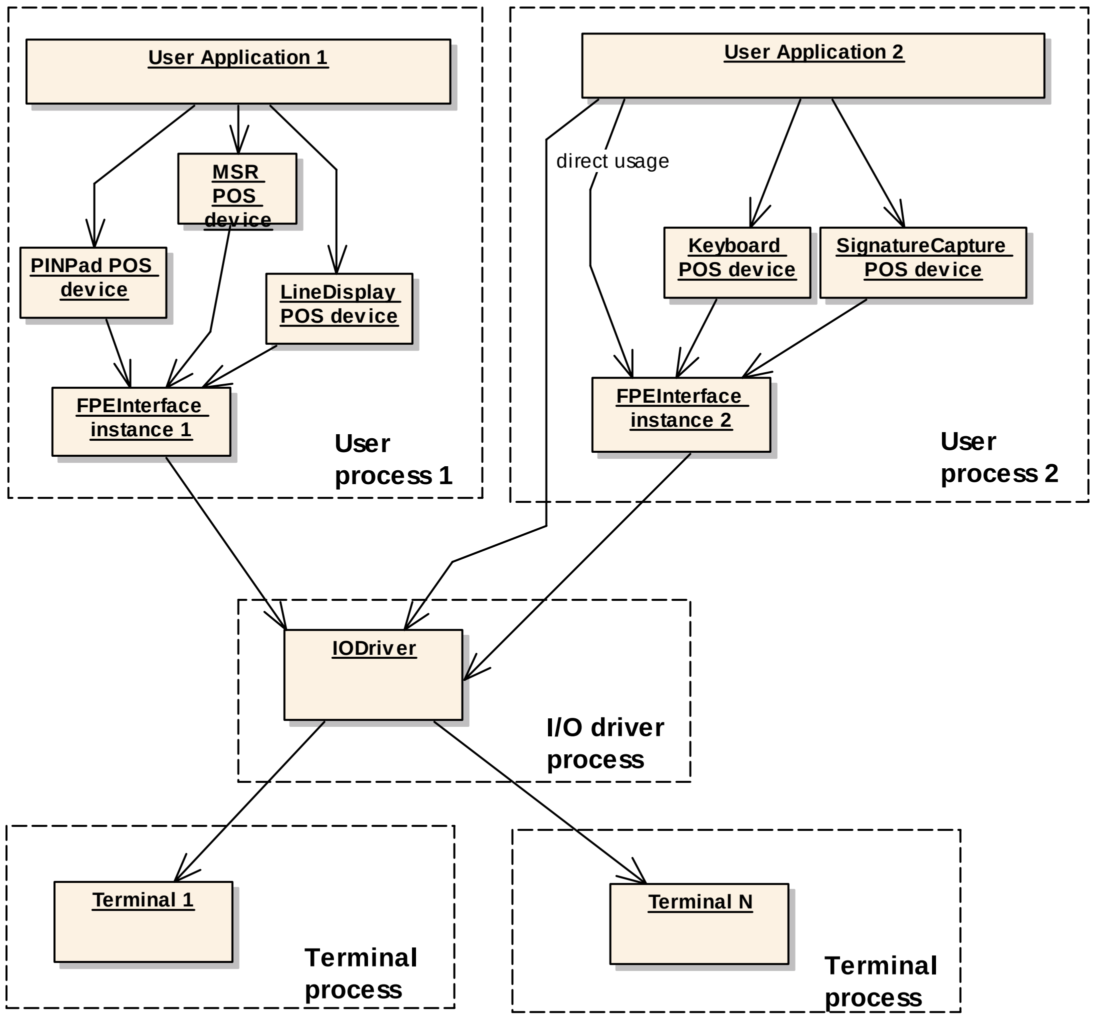


Figure 2 *Symbol JavaPOS Internal Component Model. This diagram shows an example of how JavaPOS objects can be used by two applications running simultaneously when terminals are connected via Ethernet port.*

IODriver Architecture

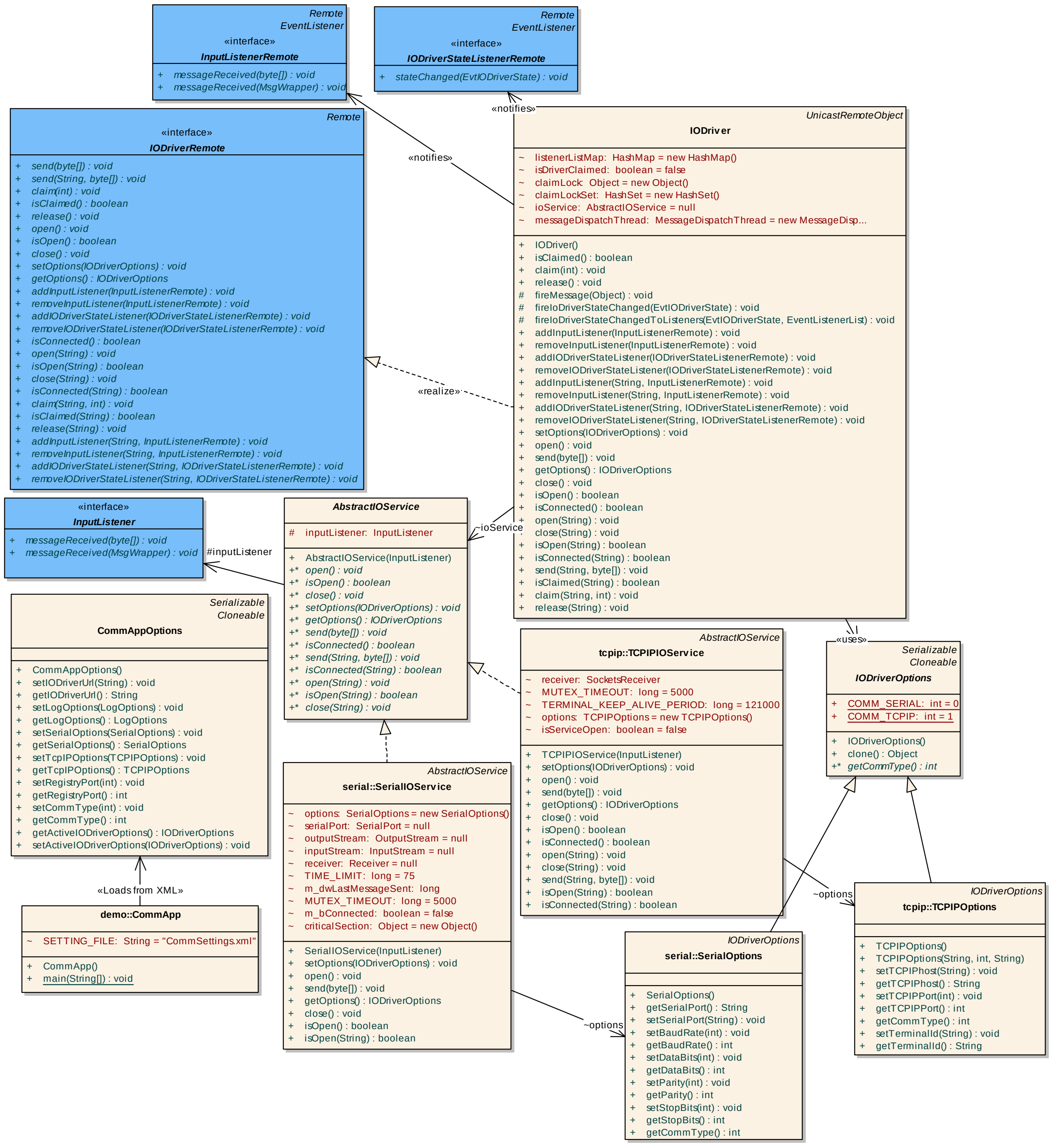

Remote
EventListener
«interface»
InputListenerRemote
+ messageReceived(byte[]) : void
+ messageReceived(MsgWrapper) : void
Remote
EventListener
«interface»
IODriverStateListenerRemote
+ stateChanged(EvtIODriverState) : void
«notifies»
«notifies»
Remote
«interface»
IODriverRemote
+ send(byte[]) : void
+ send(String, byte[]) : void
+ claim(int) : void
+ isClaimed() : boolean
+ release() : void
+ open() : void
+ isOpen() : boolean
+ close() : void
+ setOptions(IODriverOptions) : void
+ getOptions() : IODriverOptions
+ addInputListener(InputListenerRemote) : void
+ removeInputListener(InputListenerRemote) : void
+ addIODriverStateListener(IODriverStateListenerRemote) : void
+ removeIODriverStateListener(IODriverStateListenerRemote) : void
+ isConnected() : boolean
+ open(String) : void
+ isOpen(String) : boolean
+ close(String) : void
+ isConnected(String) : boolean
+ claim(String, int) : void
+ isClaimed(String) : boolean
+ release(String) : void
+ addInputListener(String, InputListenerRemote) : void
+ removeInputListener(String, InputListenerRemote) : void
+ addIODriverStateListener(String, IODriverStateListenerRemote) : void
+ removeIODriverStateListener(String, IODriverStateListenerRemote) : void
UnicastRemoteObject
IODriver
~ listenerListMap: HashMap = new HashMap()
~ isDriverClaimed: boolean = false
~ claimLock: Object = new Object()
~ claimLockSet: HashSet = new HashSet()
~ ioService: AbstractIOService = null
~ messageDispatchThread: MessageDispatchThread = new MessageDisp...
+ IODriver()
+ isClaimed() : boolean
+ claim(int) : void
+ release() : void
# fireMessage(Object) : void
# fireIoDriverStateChanged(EvtIODriverState) : void
# fireIoDriverStateChangedToListeners(EvtIODriverState, EventListenerList) : void
+ addInputListener(InputListenerRemote) : void
+ removeInputListener(InputListenerRemote) : void
+ addIODriverStateListener(IODriverStateListenerRemote) : void
+ removeIODriverStateListener(IODriverStateListenerRemote) : void
+ addInputListener(String, InputListenerRemote) : void
+ removeInputListener(String, InputListenerRemote) : void
+ addIODriverStateListener(String, IODriverStateListenerRemote) : void
+ removeIODriverStateListener(String, IODriverStateListenerRemote) : void
+ setOptions(IODriverOptions) : void
+ open() : void
+ send(byte[]) : void
+ getOptions() : IODriverOptions
+ close() : void
+ isOpen() : boolean
+ isConnected() : boolean
+ open(String) : void
+ close(String) : void
+ isOpen(String) : boolean
+ isConnected(String) : boolean
+ send(String, byte[]) : void
+ isClaimed(String) : boolean
+ claim(String, int) : void
+ release(String) : void
«realize»
~ioService
«interface»
InputListener
+ messageReceived(byte[]) : void
+ messageReceived(MsgWrapper) : void
#inputListener
AbstractIOService
# inputListener: InputListener
+ AbstractIOService(InputListener)
+* open() : void
+* isOpen() : boolean
+* close() : void
+* setOptions(IODriverOptions) : void
+* getOptions() : IODriverOptions
+* send(byte[]) : void
+* isConnected() : boolean
+* send(String, byte[]) : void
+* isConnected(String) : boolean
+* open(String) : void
+* isOpen(String) : boolean
+* close(String) : void
Serializable
Cloneable
CommAppOptions
+ CommAppOptions()
+ setIODriverUrl(String) : void
+ getIODriverUrl() : String
+ setLogOptions(LogOptions) : void
+ getLogOptions() : LogOptions
+ setSerialOptions(SerialOptions) : void
+ getSerialOptions() : SerialOptions
+ setTcpIPOptions(TCPIPOptions) : void
+ getTcpIPOptions() : TCPIPOptions
+ setRegistryPort(int) : void
+ getRegistryPort() : int
+ setCommType(int) : void
+ getCommType() : int
+ getActiveIODriverOptions() : IODriverOptions
+ setActiveIODriverOptions(IODriverOptions) : void
«Loads from XML»
demo::CommApp
~ SETTING_FILE: String = "CommSettings.xml"
+ CommApp()
+ main(String[]) : void
AbstractIOService
tcpip::TCPIPIOService
~ receiver: SocketsReceiver
~ MUTEX_TIMEOUT: long = 5000
~ TERMINAL_KEEP_ALIVE_PERIOD: long = 121000
~ options: TCPIPOptions = new TCPIPOptions()
~ isServiceOpen: boolean = false
+ TCPIPIOService(InputListener)
+ setOptions(IODriverOptions) : void
+ open() : void
+ send(byte[]) : void
+ getOptions() : IODriverOptions
+ close() : void
+ isOpen() : boolean
+ isConnected() : boolean
+ open(String) : void
+ close(String) : void
+ send(String, byte[]) : void
+ isOpen(String) : boolean
+ isConnected(String) : boolean
«uses»
Serializable
Cloneable
IODriverOptions
+ COMM_SERIAL: int = 0
+ COMM_TCPIP: int = 1
+ IODriverOptions()
+ clone() : Object
+* getCommType() : int
AbstractIOService
serial::SerialIOService
~ options: SerialOptions = new SerialOptions()
~ serialPort: SerialPort = null
~ outputStream: OutputStream = null
~ inputStream: InputStream = null
~ receiver: Receiver = null
~ TIME_LIMIT: long = 75
~ m_dwLastMessageSent: long
~ MUTEX_TIMEOUT: long = 5000
~ m_bConnected: boolean = false
~ criticalSection: Object = new Object()
+ SerialIOService(InputListener)
+ setOptions(IODriverOptions) : void
+ open() : void
+ send(byte[]) : void
+ getOptions() : IODriverOptions
+ close() : void
+ isOpen() : boolean
+ isOpen(String) : boolean
~options
IODriverOptions
tcpip::TCPIPOptions
+ TCPIPOptions()
+ TCPIPOptions(String, int, String)
+ setTCPIPHost(String) : void
+ getTCPIPHost() : String
+ setTCPIPPort(int) : void
+ getTCPIPPort() : int
+ getCommType() : int
+ setTerminalId(String) : void
+ getTerminalId() : String
~options
IODriverOptions
serial::SerialOptions
+ SerialOptions()
+ getSerialPort() : String
+ setSerialPort(String) : void
+ setBaudRate(int) : void
+ getBaudRate() : int
+ setDataBits(int) : void
+ getDataBits() : int
+ setParity(int) : void
+ getParity() : int
+ setStopBits(int) : void
+ getStopBits() : int
+ getCommType() : int

Figure 3 *IODriver class diagram.*

FPEInterface architecture

The FPETerminalInterface has the following groups of functions:

- “begin” prefixed functions are used for starting a user process
- “request” prefixed functions are used for non blocking request of a terminal data that is delivered by event after
- “get” prefixed functions similar to “request” but blocking and returning data as return value
- “set” prefixed functions are used to set some terminal parameters
- Other functions may have miscellaneous usage

See *FPETerminalInterface* JavaDoc for more detailed function descriptions.

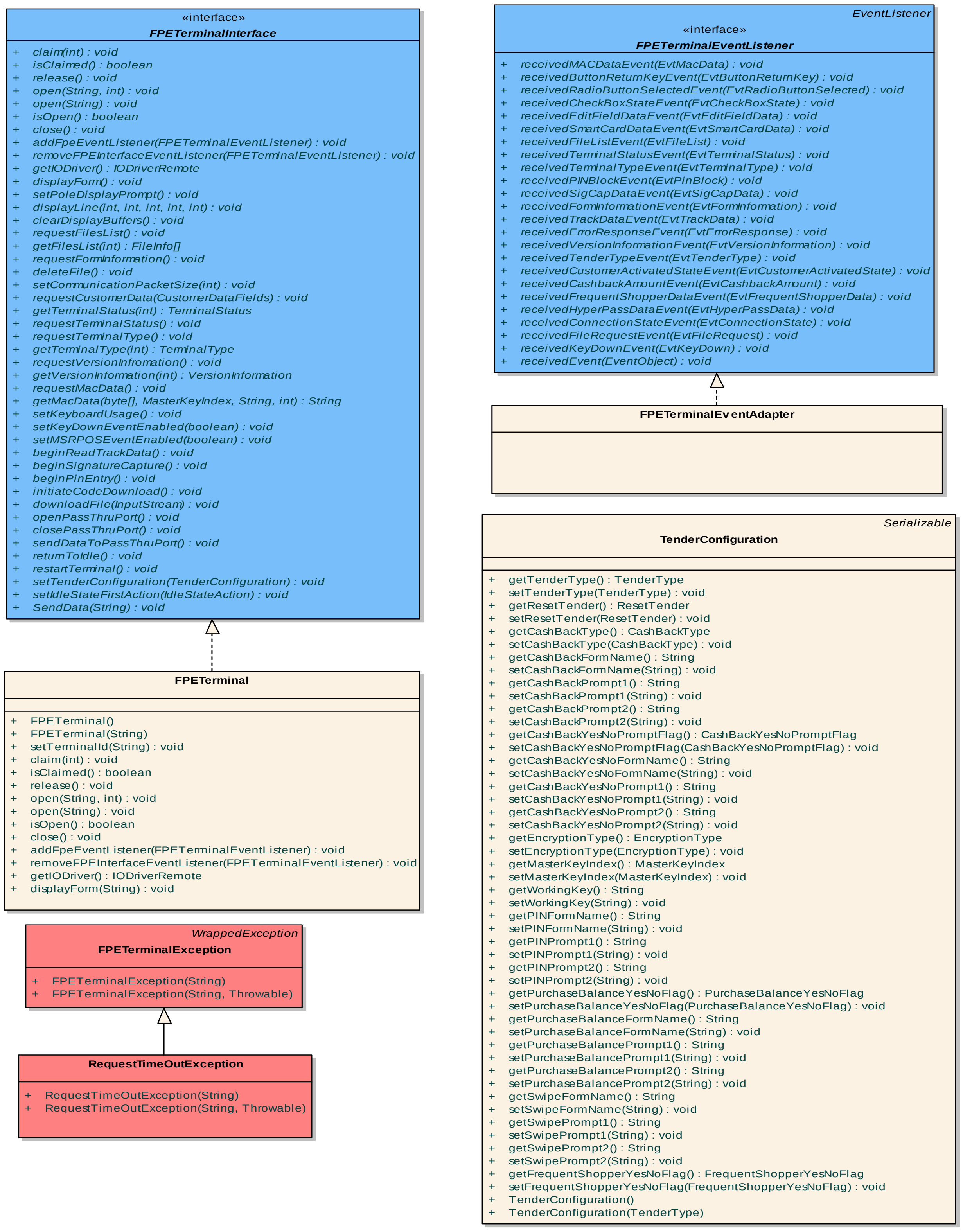

«interface»
FPETerminalInterface
+ claim(int) : void
+ isClaimed() : boolean
+ release() : void
+ open(String, int) : void
+ open(String) : void
+ isOpen() : boolean
+ close() : void
+ addFpeEventListener(FPETerminalEventListener) : void
+ removeFPEInterfaceEventListener(FPETerminalEventListener) : void
+ getIODriver() : IODriverRemote
+ displayForm() : void
+ setPoleDisplayPrompt() : void
+ displayLine(int, int, int, int, int) : void
+ clearDisplayBuffers() : void
+ requestFilesList() : void
+ getFilesList(int) : FileInfo[]
+ requestFormInformation() : void
+ deleteFile() : void
+ setCommunicationPacketSize(int) : void
+ requestCustomerData(CustomerDataFields) : void
+ getTerminalStatus(int) : TerminalStatus
+ requestTerminalStatus() : void
+ requestTerminalType() : void
+ getTerminalType(int) : TerminalType
+ requestVersionInfromation() : void
+ getVersionInformation(int) : VersionInformation
+ requestMacData() : void
+ getMacData(byte[], MasterKeyIndex, String, int) : String
+ setKeyboardUsage() : void
+ setKeyDownEventEnabled(boolean) : void
+ setMSRPOSEventEnabled(boolean) : void
+ beginReadTrackData() : void
+ beginSignatureCapture() : void
+ beginPinEntry() : void
+ initiateCodeDownload() : void
+ downloadFile(InputStream) : void
+ openPassThruPort() : void
+ closePassThruPort() : void
+ sendDataToPassThruPort() : void
+ returnToIdle() : void
+ restartTerminal() : void
+ setTenderConfiguration(TenderConfiguration) : void
+ setIdleStateFirstAction(IdleStateAction) : void
+ SendData(String) : void
EventListener
«interface»
FPETerminalEventListener
+ receivedMACDataEvent(EvtMacData) : void
+ receivedButtonReturnKeyEvent(EvtButtonReturnKey) : void
+ receivedRadioButtonSelectedEvent(EvtRadioButtonSelected) : void
+ receivedCheckBoxStateEvent(EvtCheckBoxState) : void
+ receivedEditFieldDataEvent(EvtEditFieldData) : void
+ receivedSmartCardDataEvent(EvtSmartCardData) : void
+ receivedFileListEvent(EvtFileList) : void
+ receivedTerminalStatusEvent(EvtTerminalStatus) : void
+ receivedTerminalTypeEvent(EvtTerminalType) : void
+ receivedPINBlockEvent(EvtPinBlock) : void
+ receivedSigCapDataEvent(EvtSigCapData) : void
+ receivedFormInformationEvent(EvtFormInformation) : void
+ receivedTrackDataEvent(EvtTrackData) : void
+ receivedErrorResponseEvent(EvtErrorResponse) : void
+ receivedVersionInformationEvent(EvtVersionInformation) : void
+ receivedTenderTypeEvent(EvtTenderType) : void
+ receivedCustomerActivatedStateEvent(EvtCustomerActivatedState) : void
+ receivedCashbackAmountEvent(EvtCashbackAmount) : void
+ receivedFrequentShopperDataEvent(EvtFrequentShopperData) : void
+ receivedHyperPassDataEvent(EvtHyperPassData) : void
+ receivedConnectionStateEvent(EvtConnectionState) : void
+ receivedFileRequestEvent(EvtFileRequest) : void
+ receivedKeyDownEvent(EvtKeyDown) : void
+ receivedEvent(EventObject) : void
FPETerminalEventAdapter
Serializable
TenderConfiguration
+ getTenderType() : TenderType
+ setTenderType(TenderType) : void
+ getResetTender() : ResetTender
+ setResetTender(ResetTender) : void
+ getCashBackType() : CashBackType
+ setCashBackType(CashBackType) : void
+ getCashBackFormName() : String
+ setCashBackFormName(String) : void
+ getCashBackPrompt1() : String
+ setCashBackPrompt1(String) : void
+ getCashBackPrompt2() : String
+ setCashBackPrompt2(String) : void
+ getCashBackYesNoPromptFlag() : CashBackYesNoPromptFlag
+ setCashBackYesNoPromptFlag(CashBackYesNoPromptFlag) : void
+ getCashBackYesNoFormName() : String
+ setCashBackYesNoFormName(String) : void
+ getCashBackYesNoPrompt1() : String
+ setCashBackYesNoPrompt1(String) : void
+ getCashBackYesNoPrompt2() : String
+ setCashBackYesNoPrompt2(String) : void
+ getEncryptionType() : EncryptionType
+ setEncryptionType(EncryptionType) : void
+ getMasterKeyIndex() : MasterKeyIndex
+ setMasterKeyIndex(MasterKeyIndex) : void
+ getWorkingKey() : String
+ setWorkingKey(String) : void
+ getPINFormName() : String
+ setPINFormName(String) : void
+ getPINPrompt1() : String
+ setPINPrompt1(String) : void
+ getPINPrompt2() : String
+ setPINPrompt2(String) : void
+ getPurchaseBalanceYesNoFlag() : PurchaseBalanceYesNoFlag
+ setPurchaseBalanceYesNoFlag(PurchaseBalanceYesNoFlag) : void
+ getPurchaseBalanceFormName() : String
+ setPurchaseBalanceFormName(String) : void
+ getPurchaseBalancePrompt1() : String
+ setPurchaseBalancePrompt1(String) : void
+ getPurchaseBalancePrompt2() : String
+ setPurchaseBalancePrompt2(String) : void
+ getSwipeFormName() : String
+ setSwipeFormName(String) : void
+ getSwipePrompt1() : String
+ setSwipePrompt1(String) : void
+ getSwipePrompt2() : String
+ setSwipePrompt2(String) : void
+ getFrequentShopperYesNoFlag() : FrequentShopperYesNoFlag
+ setFrequentShopperYesNoFlag(FrequentShopperYesNoFlag) : void
+ TenderConfiguration()
+ TenderConfiguration(TenderType)
FPETerminal
+ FPETerminal()
+ FPETerminal(String)
+ setTerminalId(String) : void
+ claim(int) : void
+ isClaimed() : boolean
+ release() : void
+ open(String, int) : void
+ open(String) : void
+ isOpen() : boolean
+ close() : void
+ addFpeEventListener(FPETerminalEventListener) : void
+ removeFPEInterfaceEventListener(FPETerminalEventListener) : void
+ getIODriver() : IODriverRemote
+ displayForm(String) : void
WrappedException
FPETerminalException
+ FPETerminalException(String)
+ FPETerminalException(String, Throwable)
RequestTimeOutException
+ RequestTimeOutException(String)
+ RequestTimeOutException(String, Throwable)

Figure 4 *FPETerminalInterface class diagram. The internal structure is more complicated. Here are the most important classes are presented.*

JavaPOS Service Architecture

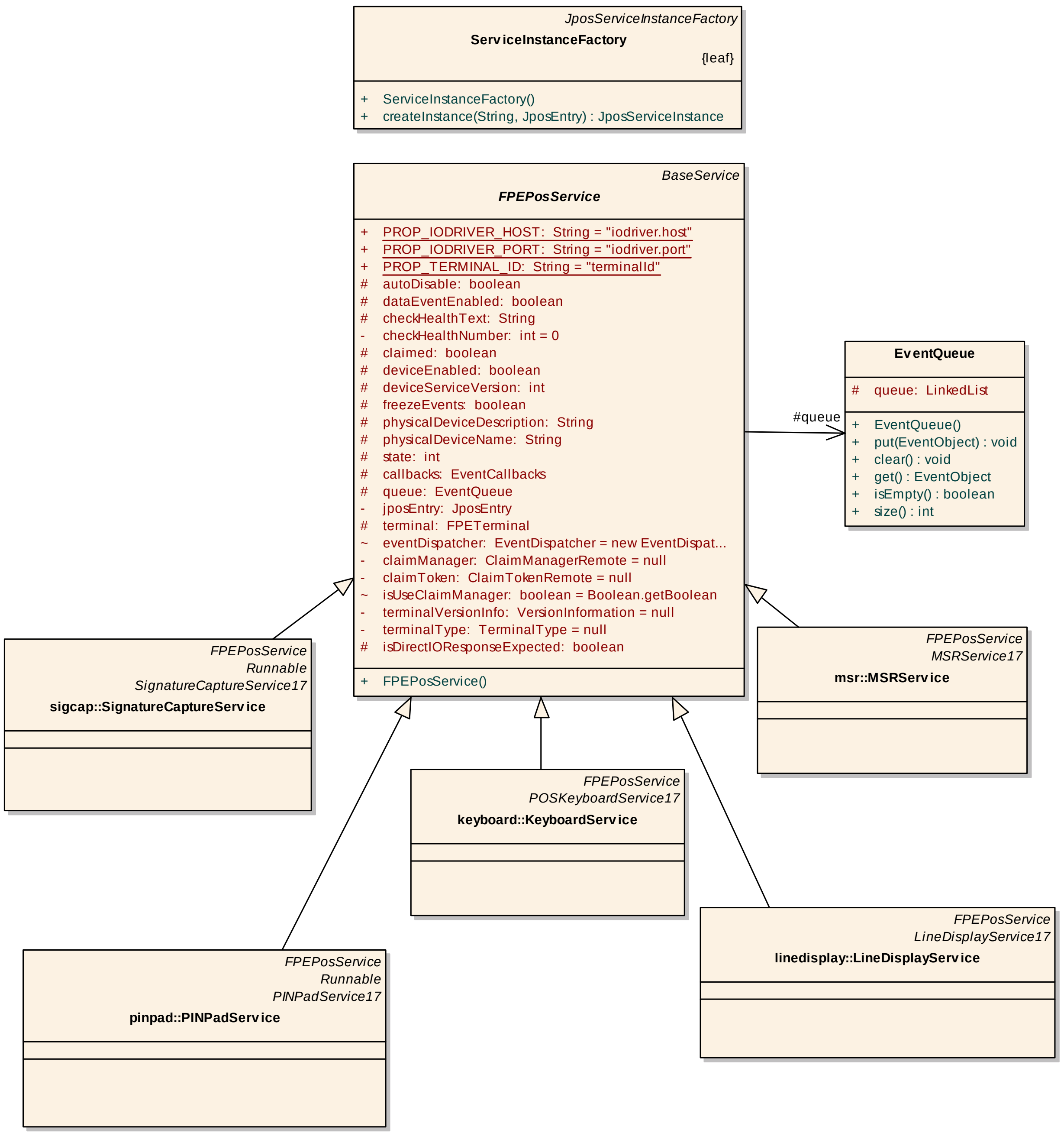

JposServiceInstanceFactory
ServiceInstanceFactory
{leaf}
+ ServiceInstanceFactory()
+ createInstance(String, JposEntry) : JposServiceInstance
BaseService
FPEPosService
+ PROP_IODRIVER_HOST: String = "iodriver.host"
+ PROP_IODRIVER_PORT: String = "iodriver.port"
+ PROP_TERMINAL_ID: String = "terminalId"
# autoDisable: boolean
# dataEventEnabled: boolean
# checkHealthText: String
- checkHealthNumber: int = 0
# claimed: boolean
# deviceEnabled: boolean
# deviceServiceVersion: int
# freezeEvents: boolean
# physicalDeviceDescription: String
# physicalDeviceName: String
# state: int
# callbacks: EventCallbacks
# queue: EventQueue
- jposEntry: JposEntry
# terminal: FPETerminal
~ eventDispatcher: EventDispatcher = new EventDispat...
- claimManager: ClaimManagerRemote = null
- claimToken: ClaimTokenRemote = null
~ isUseClaimManager: boolean = Boolean.getBoolean
- terminalVersionInfo: VersionInformation = null
- terminalType: TerminalType = null
# isDirectIOResponseExpected: boolean
+ FPEPosService()
#queue
EventQueue
# queue: LinkedList
+ EventQueue()
+ put(EventObject) : void
+ clear() : void
+ get() : EventObject
+ isEmpty() : boolean
+ size() : int
FPEPosService
Runnable
SignatureCaptureService17
sigcap::SignatureCaptureService
FPEPosService
MSRService17
msr::MSRService
FPEPosService
POSKeyboardService17
keyboard::KeyboardService
FPEPosService
LineDisplayService17
linedisplay::LineDisplayService
FPEPosService
Runnable
PINPadService17
pinpad::PINPadService

Figure 5 *JavaPOS device service diagram*

DEVICE SHARING MODEL

The system assumes that multiple objects in different applications can communicate to terminal simultaneously. This means that Input/Output operations must be somehow synchronized. The synchronization is performed on two levels: on **IODriver** level and on **FPEInterface** level. The other components as POS objects and user application can control synchronization trough these objects. There is also special synchronization level on the JavaPOS layer that synchronizes different types of POS device objects.

IODriver Level Synchronization

The IODriver functional interface provides three synchronization control functions:

claim();
claim(String terminalId);
isClaimed();
isClaimed(String terminalId);
release();
release(String terminalId);

These functions help to control access to **send** function performing sending of FPE data packet to terminal. Before using **send** function the user component must call **claim** function to mark start of output transaction. After **claim** call completes successfully the user component is allowed to call **send** function (possibly several times) and when necessary data is transmitted the **release** function should be called to mark end of output transaction. In case if one component has started transaction while another tries to start another transaction the second component receives an exception performing **claim** function call. In this case the component is not allowed to make data transmission until the first component finishes.
IO driver does not deny calling of **send** function without previous **claim** operation and it is the user responsibility to synchronize transactions in this case. The **send** function has atomic behavior, thus any new **send** call is not starting data transmission until the previous is finished.

FPEInterface Level Synchronization

The **FPEInterface** performs synchronization on each method basis and also has the possibility to demarcate transactions.
By default each method begins and finishes a separate transaction by calling **claim** and **release** methods of **IODriver** before and after FPE command transmission. Each such method also made thread safe.
In case it is necessary to perform multiple **FPEInterface** method calls in guaranteed transactional sequence the **FPEInterface** provides the same methods as **IODriver**:

claim();
isClaimed();
release();

The methods use **IODriver's** method calls inside having one difference that they are reentrant inside one **FPEInterface** object. This means that it's allowed to call **claim** method multiple times without calling release for one **FPEInterface** instance and it won't throw an exception in case if first call completed successfully.

JavaPOS Device Sharing Model

WARNING: *This section is deprecated. It describes sharing model based on the ClaimManager that is switched off by default.*

The main problem with Unified POS standard is that it assumes that each separate POS component has its own separate hardware device. In the case of Symbol terminals, there is one hardware device that includes several device features inside. Each of the OPOS devices (PINPad, MSR, SignatureCapture, Keyboard and Line Display) is declared as “exclusive use” device in the UnifiedPOS standard. The supplied solution is to deny claiming some POS devices depending on current system state. The table below shows supplied default exclusion matrix. The “X” is put in the cells where two devices (one in the row and one in the column) cannot be claimed simultaneously.

	PIN pad	**Line Display**	**Signature Capture**	**MSR**	**Keyboard**
PIN pad	**X**	**X**	**X**	**X**	**X**
Line Display	**X**	**X**			
Signature Capture	**X**		**X**		
MSR	**X**			**X**	
Keyboard	**X**				**X**

For making **claim** operation each POS device ask a singleton **ClaimManager** object if the device is allowed to make **claim**. The **ClaimManager** can be a local in-process object or remote shared object that manages POS devices of different applications. It has exclusion matrix inside that can be reconfigured in case if exclusion policy should be changed.

WARNING: By default, JPOS services do not use **ClaimManager**. To switch usage on, it is necessary to set the **com.hypercom.fpe.jpos.isUseClaimManager** system property to true by specifying it in client java application parameter:

```
java -Dcom.hypercom.fpe.jpos.isUseClaimManager=true ….
```

JCL Configuration

POS controls read additional parameters from JCL module that provides visual configuring tools of configuration parameters and stores parameter data in XML format. The parameters supported by each implemented service are described in “Configurable static parameters” section of each service description. Sample configuration for Line Display, Signature Capture and MSR category devices is listed below (please refer to Appendix B for sample configuration of all supported device categories).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE JposEntries PUBLIC "-//JavaPOS//DTD//EN" "lib/jpos/res/jcl.dtd">
<JposEntries>
<!--Saved by JavaPOS jpos.config/loader (JCL) version 2.1.0-RC3 on 1/26/04 8:20 AM-->

 <JposEntry logicalName="LineDisplay">
        <creation factoryClass="com.hypercom.fpe.jpos.ServiceInstanceFactory"
serviceClass="com.hypercom.fpe.jpos.linedisplay.LineDisplayService"/>
        <vendor name="Hypercom Corporation" url="http://www.hypercom.com"/>
        <jpos category="LineDisplay" version="1.7"/>
        <product description="Hypercom JavaPOS for FPE terminal" name="Hypercom JavaPOS
for FPE terminal" url="http://www.hypercom.com"/>
        <!--Other non JavaPOS required property (mostly vendor properties and bus
specific properties i.e. RS232 )-->
        <prop name="deviceBus" type="String" value="Unknown"/>
        <prop name="foregroundColor" type="Integer" value="0"/>
        <prop name="backgroundColor" type="Integer" value="255"/>
        <prop name="fontSize" type="Integer" value="0"/>
</JposEntry>


    <JposEntry logicalName="MSR">
        <creation factoryClass="com.hypercom.fpe.jpos.ServiceInstanceFactory"
serviceClass="com.hypercom.fpe.jpos.msr.MSRService"/>
        <vendor name="Hypercom Corporation" url="http://www.hypercom.com"/>
        <jpos category="MSR" version="1.7"/>
        <product description="Hypercom JavaPOS for FPE terminal" name="Hypercom JavaPOS
for FPE terminal" url="http://www.hypercom.com"/>

        <!--Other non JavaPOS required property (mostly vendor properties and bus
specific properties i.e. RS232 )-->
        <prop name="swipeFormName" type="String" value="SWIPEFRM"/>
        <prop name="deviceBus" type="String" value="Unknown"/>
        <prop name="gloablPromptString2" type="String" value="OR USE SPEEDPASS"/>
        <prop name="gloablPromptString1" type="String" value="PLEASE SLIDE YOUR CARD"/>
</JposEntry>


    <JposEntry logicalName="SignatureCapture">
        <creation factoryClass="com.hypercom.fpe.jpos.ServiceInstanceFactory"
serviceClass="com.hypercom.fpe.jpos.sigcap.SignatureCaptureService"/>
        <vendor name="Hypercom Corporation" url="http://www.hypercom.com"/>
        <jpos category="LineDisplay" version="1.7"/>
        <product description="Hypercom JavaPOS for FPE terminal" name="Hypercom JavaPOS
for FPE terminal" url="http://www.hypercom.com"/>

  <!--Other non JavaPOS required property (mostly vendor properties and bus specific
properties i.e. RS232 )-->
        <prop name="deviceBus" type="String" value="Unknown"/>
        <prop name="resolution" type="String" value="High"/>
        <prop name="globalPromptNumber" type="Integer" value="1"/>
        <prop name="waitFormName" type="String" value="WAITFRM"/>
        <prop name="globalPromptString" type="String" value="Please Sign Below"/>
        <prop name="penUpTimeout" type="Integer" value="9"/>
</JposEntry>
…
</JposEntries>
```

COMMON PROPERTIES

These are properties common to all supported services.

Property	Value	Description
AutoDisable		Supported in MSR, PINPad and SignatureCapture
CapPowerReporting: *int32* { read-only } 1.3 open	Always PR_NONE	There is no reliable way to detect if the ICE6000/5500 is powered on or off
CheckHealthText: *string* { read-only } 1.3 open		Initially has “” value. Reports either “Internal HCheck: Successful” when **checkHealth** has successful result and “Internal HCheck: Error <description>” if **checkHealth** failes. The <description> contains error description.
Claimed: *boolean* { read-only } 1.3 open	True or False	Simple local variable of JavaPOS class determining “claimed” state. Updated when “claim” or “release” methods are called.
DataCount: *int32* { read-only } 1.3 open	Initially Zero	Simple local variable. Number of unhandled events in queue
DataEventEnabled: *boolean* { read-write } 1.3 open	True or False	Simple local variable
DeviceEnabled: *boolean* { read-write } 1.3 open & claim	True or False	Simple local variable
FreezeEvents: *boolean* { read-write } 1.3 open	True or False	Simple local variable
OutputID: *int32* { read-only } 1.3	Not Supported	Not Supported
PowerNotify: *int32* { read-write } 1.3 open	Not supported	Not Supported
PowerState: *int32* { read-only } 1.3 open	Not supported	Not supported
State: *int32* { read-only } 1.3 –	JPOS_S_CLOSE, JPOS_S_IDLE	If open method was not called before than it has JPOS_S_CLOSE value. In all other cases device is in JPOS_S_IDLE state.
DeviceControlDescription: *string* { read-only } 1.3 –		This is defined in JavaPOS API sources
DeviceControlVersion: *int32* { read-only } 1.3 –		This is defined in JavaPOS API sources
DeviceServiceDescription: *string* { read-only } 1.3 open PINPad, MSR etc	Hypercom <name> JavaPOS service	The <name> has values PINPad, MSR, LineDisplay, SignatureCapture, KeyBoard depending on component.
DeviceServiceVersion: *int32* { read-only } 1.3 open;	1007000	Corresponds to 1.7 version of Unified POS
PhysicalDeviceDescription: *string* { read-only } 1.3 open getTerminalInfo, getVersionInfo	Holds terminal type and OS version description	The property value is filled inside “open” method. It requests terminal information by issuing “Terminal Type 'T'” and Version Information 'F' FPE commands

Property	Value	Description
PhysicalDeviceName: *string* { read-only } 1.3 open getTerminalInfo, getVersionInfo	Terminal name ICE6000 or ICE5500 or other compatible terminal	The property value is filled inside "open" method. It requests terminal information by issuing "Terminal Type 'T'" FPE command.

COMMON METHODS

Methods common to all supported services

Method	Description
checkHealth (level: *int32*): void { raises-exception, use after open, claim, enable }	Supported "level" value is CH_INTERNAL. When the method is called it passes "Terminal Status 'S'" FPE command and makes sure it receives valid response. Later it is possible to add CH_INTERACTIVE level support that shows FPEDemo modal dialog where the user can test functionality interactively.
claim (timeout: *int32*): void { raises-exception }	Calls **ClaimManager.claim(<device_type>)** method. In case the call does not throw an exception device can continue claim initializations, otherwise **claim** forbidden in the current system state.
clearInput ( ): void { raises-exception }	Clears input even queue as it is written in the Unified POS specification.
clearOutput ( ): void { raises-exception }	Not used, because all output methods are synchronous
close ( ): void { raises-exception }	Works as specified in UnifiedPOS
directIO (command: *int32*, inout data: *int32*, inout obj: *object*): void { raises-exception }	This method delegates its job to IODriver.send method. ✓**Note:** In case of jpos.MSR, this method can be used to obtain Hypercom specific MSR service property *.
open (logicalDeviceName: *string*): void { raises-exception }	Works as specified in UnifiedPOS
release ( ): void { raises-exception }	Calls **ClaimManager.release(<device_type>)** method

* Hypercom JavaPOS service library v1.1.5 introduces following specific feature of directIO() method implementation for MSR service:
If directIO() method called on MSR control is passed "data" parameter equal to "TS", then the first element of "data" (data[0]) INOUT method parameter after method execution will contain ASCII numeric code of TrackDataSource as it is described in FPE Interface specification ([3]), e.g. '0' - TRACK_READER, '1' - RDIF_EXPRESS_PAY, '2' - RFID_PAY_PASS, '3' - RFID_VISA_WAVE '4' - RFID_UNKNOWN_APP. . See Appendix D for MSR control usage sample code.

PIN PAD

Device type:	Exclusive
Java Class:	jpos.PINPad
Service Class:	com.hypercom.jpos.pinpad.PINPadService
Service Factory Class:	com.hypercom.jpos.ServiceInstanceFactory
JCL Entry name:	PINPad

Configurable Static Parameters

Parameters that are specified in the XML file of JCL

Property	Default Value	Description
masterKeyIndex	1	(1-9) Used master key index
workingKey		Hexadecimal 16 byte value, Working key to use
pinFormName	PINFRM	The form name should be used during PIN entry operation.
promptFormName	PROMPTS	Form is used to show prompts when "Prompt" property is changed
globalPromptNumber	-1	-1 – Means that no prompt will be set (1, 2, 3, 4) – values are allowed
globalPromptString	null	Text displayed in the prompt
terminalId	null	Should be valid terminal serial number (12 digit number). If property value is not specified, then terminal id is taken from current IODriverOptions. Taken into account only for communication via Ethernet port
creditTransactionKey	ALL	Function key events to be sent to ECR. This is optional property. If it's not specified, the default the value "ALL" is assign to it. If an other value(e.g. "FuncKey1") is assigned to the property , then ECR will receive notification when it is pressed and all the other ones will be ignored

Methods

Method	Description
beginEFTTransaction (PINPadSystem: ***string,*** **transactionHost:** ***int32*** **):** void { raises-exception, use after open-claim-enable }	Nothing to do in our case. Simple switches on transactionBegun flag.
enablePINEntry (): void { raises-exception, use after beginEFTTransaction);	Executes Form Request that shows PIN entry form. Sets Prompt property to PPAD_MSG_ENTERPIN (1) value.
computeMAC (inMsg: ***string,*** **outMsg:** ***object*** **):** void { raises-exception, use after beginEFTTransaction)	Calls **FPEInterface.requestMacData** function and waits for MAC data event back.

Method	Description
endEFTTransaction (completionCode: ***int32*****):** void { raises-exception, use after beginEFTTransaction }	Nothing to do in our case. Simple switches off transactionBegun flag.
updateKey (keyNum: ***int32*****, key:** ***string*****):** void { raises-exception, use after beginEFTTransaction }	Simply sets the local variable holding working key value. The working key value will be used when calling **FPEInterface.beginPinEntry** function. Values of workingKey and MasterKeyIndex are updated in JCL file, so they will be remembered for the next sessions
verifyMAC (message: ***string*** **):** void { raises-exception, use after beginEFTTransaction }	Throws exception. Not supported function.

Properties

Property	Value	Description
CapDisplay: *int32* { read-only } 1.3 open	PPAD_DISP_RESTRICTED_LIST	PIN pad will be able to show only predefined messages that are set to **Prompt** property.
CapKeyboard: *boolean* { read-only } 1.3 open	True	Keyboard control is available separately
CapLanguage: *int32* { read-only } 1.3 open	PPAD_LANG_ONE	Support only one language EN,US
CapMACCalculation: *boolean* { read-only } 1.3 open	True	
CapTone: *boolean* { read-only } 1.3 open	False	We don’t provide control of key tones
AccountNumber: *string* { read-write, access after open }	Account number	Simple local variable that is used in beginEFTTransaction
AdditionalSecurityInformation: *string* { read-only } 1.3 open	PIN Pad sequence number	Used in only DUKPT encryption mode. In all other cases is empty.
Amount: *int32* { read-write } 1.3 open	Any int	Simple local variable
AvailableLanguagesList: *string* { read-only } 1.3 open	**EN,US;**	Only one language is supported
AvailablePromptsList: *string* { read-only } 1.3 open		Support for all prompts described in Unified POS.
EncryptedPIN: *string* { read-only } 1.3 open	PIN block	Contains PIN block available after EFTTransaction.
MaximumPINLength: *int32* { read-write } 1.3 open	Default: 7	This actually depends on the form in the form builder
MerchantID: *string* { read-write } 1.3 open	Any string	Simple Local variable
MinimumPINLength: *int32* { read-write } 1.3 open	Default: 4	This actually depends on the form in the form builder

Property	Value	Description
PINEntryEnabled: *boolean* { read-only } 1.3 open	True or False	True if enablePINEntry was started
Prompt: *int32* { read-write } 1.3 open	One of prompt numbers described in Unified POS	If this property is set when device is enabled the terminal displays corresponding form with prompt text.
PromptLanguage: *nls* { read-write } 1.3 open	One of language codes specified in the **LanguagesList** property.	Changes **Prompt** property language
TerminalID: *string* { read-write } 1.3 open	Any string	Simple local variable
Track1Data: *binary* { read-write } 1.3 open	Any	Simple local variable
Track2Data: *binary* { read-write } 1.3 open	Any	Simple local variable
Track3Data: *binary* { read-write } 1.3 open	Any	Simple local variable
Track4Data: *binary* { read-write } 1.5 open	Any	Simple local variable
TransactionType: *string* { read-write } 1.3 open	Any	Simple local variable

LINEDISPLAY

Device type:	Exclusive
Java Class:	jpos.LineDisplay
Service Class:	com.hypercom.jpos.linedisplay.LineDisplayService
Service Factory Class:	com.hypercom.jpos.ServiceInstanceFactory
JCL Entry name:	LineDisplay

Configurable Static Parameters

Parameters that are specified in the XML file of JCL

Property	Default value	Description
backgroundColor	255	(0-255) LineDisplay font background color
foregroundColor	0	(0 – 255) LineDisplay font foreground color
fontSize	0	(0, 1, 2, 3, 4) Font size
terminalId	null	Should be valid terminal serial number (12 digit number). If property value is not specified, then terminal id is taken from current IODriverOptions. Taken into account only for communication via Ethernet port

Methods

Method	Description
clearDescriptors ( ): void { raises-exception, use after open-claim-enable }	Always throws E_ILLEGAL The device does not support descriptors.
clearText ( ): void { raises-exception, use after open-claim-enable }	Calls FPETerminalInterface.clearDisplayBuffers function (**FPE:** Clear All Display Lines 'N')
createWindow (viewportRow: ***int32*****, viewportColumn:** ***int32*****, viewportHeight:** ***int32*****, viewportWidth:** ***int32*****, windowHeight:** ***int32*****, windowWidth:** ***int32*** **):** void { raises-exception, use after open-claim-enable }	Always throws E_ILLEGAL The device does not support windows.
defineGlyph (glyphCode: ***int32*****, glyph:** ***binary*** **):** void { raises-exception, use after open-claim-enable }	Throws E_ILLEGAL The device does not support glyph.
destroyWindow ( ): void { raises-exception, use after open-claim-enable }	Always throws E_ILLEGAL The device does not support windows.
displayBitmap (fileName:***string*****,width:*****int32*****,alignmentX:*****int32*****,alignmentY:*****int32*** **):** void { raises-exception, use after open-claim-enable }	Always throws E_ILLEGAL. Bitmap displaying is not supported.
displayText (data: ***string*****, attribute:** ***int32*** **):** void { raises-exception, use after open-claim-enable }	Calls FPETerminalInterface.displayLine. Uses **CursorRow** and **CursorPos** as X,Y Supports only DISP_DT_NORMAL and DISP_DT_BLINK attributes (**FPE:** UpdateDisplay ‘M’)
displayTextAt (row: ***int32*****, column:** ***int32*****, data:** ***string*****, attribute:** ***int32*** **):** void { raises-exception, use after open-claim-enable }	Calls FPETerminalInterface.displayLine. Supports only DISP_DT_NORMAL and DISP_DT_BLINK attributes (**FPE:** UpdateDisplay ‘M’)

Method	Description
readCharacterAtCursor (inout cursorData: ***int32*** **):** void { raises-exception, use after open-claim-enable }	Throws E_ILLEGAL The device does not support reading.
refreshWindow (window: ***int32*** **):** void { raises-exception, use after open-claim-enable }	Always throws E_ILLEGAL The device does not support windows.
scrollText (direction: ***int32*****, units:** ***int32*** **):** void { raises-exception, use after open-claim-enable }	Always throws E_ILLEGAL. Scrolling is not supported.
setBitmap (bitmapNumber: ***int32*****, fileName:** ***string*****, width:** ***int32*****, alignmentX:** ***int32*****, alignmentY:** ***int32*** **):** void { raises-exception, use after open-claim-enable }	Always throws E_ILLEGAL. Bitmaps are not supported.
setDescriptor (descriptor: ***int32*****, attribute:** ***int32*** **):** void { raises-exception, use after open-claim-enable }	Always throws E_ILLEGAL. Descriptors are not supported.

Properties

Property	Value	Description
CapBlink: *int32* { read-only } 1.0 open	DISP_CB_BLINKEACH	
CapBitmap: *boolean* { read-only } 1.7 open	False	
CapBlinkRate: *boolean* { read-only } 1.6 open	True	
CapBrightness: *boolean* { read-only } 1.0 open	False	
CapCharacterSet: *int32* { read-only } 1.0 open	DISP_CCS_ASCII	
CapCursorType: *int32* { read-only } 1.6 open	DISP_CCT_NONE	
CapCustomGlyph: *boolean* { read-only } 1.6 open	False	
CapDescriptors: *boolean* { read-only } 1.0 open	False	
CapHMarquee: *boolean* { read-only } 1.0 open	False	
CapICharWait: *boolean* { read-only } 1.0 open	False	
CapMapCharacterSet: *boolean* { read-only } 1.7 open	False	
CapReadBack: *int32* { read-only } 1.6 open	DISP_CRB_NONE	
CapReverse: *int32* { read-only } 1.6 open	DISP_CR_NONE	
CapScreenMode: *boolean* { read-only } 1.7 open	False	
CapVMarquee: *boolean* { read-only } 1.0 open	False	
BlinkRate: *int32* { read-write } 1.6 open	Default: 990	
CharacterSet: *int32* { read-write } 1.0 open, claim, & enable	DISP_CS_ASCII	Only one character set is supported

Property	Value	Description
CharacterSetList: *string* { read-only } 1.0 open	"998,999"	DISP_CS_ASCII and DISP_CS_ANSI
Columns: *int32* { read-only } 1.0 open	**DeviceColumns**	
CurrentWindow: *int32* { read-write } 1.0 open	0	Always zero. Throws E_ILLEGAL when attempting to change to non-zero value
CursorColumn: *int32* { read-write } 1.0 open	0< CursorColumn < Columns	Updated each time new line is displayed if **CursorUpdate == True**
CursorRow: *int32* { read-write } 1.0 open	0< CursorRow < Rows	Updated each time new line is displayed if **CursorUpdate == True**
CursorType: *int32* { read-write } 1.6 open	DISP_CT_NONE	
CursorUpdate: *boolean* { read-write } 1.0 open	True or False	If True, the **CursorRow** and **CursorColumn** are updated
CustomGlyphList: *string* { read-only } 1.6 open	""	Empty string. Not supported.
DeviceBrightness: *int32* { read-write } 1.0 open, claim, & enable	100	Allowed to be changed, but does not affect real brightness
DeviceColumns: *int32* { read-only } 1.0 open	43 (ICE6000)	
DeviceDescriptors: *int32* { read-only } 1.0 open	0	
DeviceRows: *int32* { read-only } 1.0 open	16 (ICE6000)	
DeviceWindows: *int32* { read-only } 1.0 open	0	Only device window is supported
GlyphHeight: *int32* { read-only } 1.6 open	0	Not supported
GlyphWidth: *int32* { read-only } 1.6 open	0	Not supported
InterCharacterWait: *int32* { read-write } 1.0 open	**Default:** 0	Allowed to be changed, but does not affect terminal
MapCharacterSet: *boolean* { read-write } 1.7 open	**Default:** False	Allowed to be changed, but does not affect behavior
MarqueeFormat: *int32* { read-write } 1.0 open	**Default:** DISP_MF_PLACE	Allowed to be changed, but does not affect behavior
MarqueeRepeatWait: *int32* { read-write } 1.0 open	**Default: 0**	Allowed to be changed, but does not affect behavior
MarqueeType: *int32* { read-write } 1.0 open	**Default:** DISP_MT_NONE	Throws E_ILLEGAL **w**hen attempted to be changed

Property	Value	Description
MarqueeUnitWait: *int32* { read-write } 1.0 open	Default: 0	Allowed to be changed, but does not affect behavior
MaximumX: *int32* { read-only } 1.7 open	0	
MaximumY: *int32* { read-only } 1.7 open	0	
Rows: *int32* { read-only } 1.0 open	DeviceRows	
ScreenMode: *int32* { read-write } 1.7 open & claim	0	Throws E_ILLEGAL if not 0 is attempted to be written
ScreenModeList: *string* { read-only } 1.7 open	15x42 (ICE6000)	

MSR

Device type:	Exclusive
Java Class:	jpos.MSR
Service Class:	com.hypercom.jpos.msr.MSRService
Service Factory Class:	com.hypercom.jpos.ServiceInstanceFactory
JCL Entry name:	MSR

Configurable static parameters

Parameters that are specified in the XML file of JCL

Property	Default Value	Description
terminalId	null	Should be valid terminal serial number (12 digit number). If property value is not specified, then terminal id is taken from current IODriverOptions. Taken into account only for communication via Ethernet port

Properties

Property	Value	Description
CapISO: *boolean* { read-only } 1.0 open	True	
CapJISOne: *boolean* { read-only } 1.0 open	False	
CapJISTwo: *boolean* { read-only } 1.0 open	False	
CapTransmitSentinels: *boolean* { read-only } 1.5 open	False	
AccountNumber: *string* { read-only } 1.0 open	Filled when card is swiped	Arrives with card data event.
DecodeData: *boolean* { read-write } 1.0 open		As in UnifiedPOS specification
ErrorReportingType: *int32* { read-write } 1.2 open		
ExpirationDate: *string* { read-only } 1.0 open		
FirstName: *string* { read-only } 1.0 open		
MiddleInitial: *string* { read-only } 1.0 open		
ParseDecodeData: *boolean* { read-write } 1.0 open		As in UnifiedPOS specification
ServiceCode: *string* { read-only } 1.0 open		
Suffix: *string* { read-only } 1.0 open		
Surname: *string* { read-only } 1.0 open		
Title: *string* { read-only } 1.0 open		
Track1Data: *binary* { read-only } 1.0 open		

Property	Value	Description
Track1DiscretionaryData: *binary* { read-only } 1.0 open		
Track2Data: *binary* { read-only } 1.0 open		
Track2DiscretionaryData: *binary* { read-only } 1.0 open		
Track3Data: *binary* { read-only } 1.0 open		
Track4Data: *binary* { read-only } 1.5 open		
TracksToRead: *int32* { read-write } 1.0 open		As in UnifiedPOS specification
TransmitSentinels: *boolean* { read-write } 1.5 open		

KEYBOARD

Device type:	Exclusive
Java Class:	jpos.Keyboard
Service Class:	com.hypercom.jpos.keyboard.KeyboardService
Service Factory Class:	com.hypercom.jpos.ServiceInstanceFactory
JCL Entry name:	Keyboard

Configurable static parameters

Parameters that are specified in the XML file of JCL

Property	Default Value	Description
terminalId	null	Should be valid terminal serial number (12 digit number). If property value is not specified, then terminal id is taken from current IODriverOptions. Taken into account only for communication via Ethernet port

Properties

Property	Value	Description
CapKeyUp: *boolean* { read-only } 1.2 open	False	
EventTypes: *int32* { read-write } 1.2 open	KBD_ET_DOWN	The KBD_ET_UP is not allowed
POSKeyData: *int32* { read-only } 1.1 open		Key code filled by last key event
POSKeyEventType: *int32* { read-only } 1.2 open	**KBD_KET_KEYDOWN, KBD_KET_KEYUP**	

SIGNATURE CAPTURE

Device type:	Exclusive
Java Class:	jpos.SignatureCapture
Service Class:	com.hypercom.jpos.sigcap.SignatureCaptureService
Service Factory Class:	com.hypercom.jpos. ServiceInstanceFactory
JCL Entry name:	SignatureCapture

Configurable Static Parameters

Property	Default Value	Description
resolution	High	“High” or “Low” resolution string value High – 1024x1024 Low – 640 - 480
penUpTimeout	9	(1-9) Pen Up timeout in seconds. After this time signature is automatically transferred to host
globalPromptNumber	-1	-1 – Means that no prompt will be set (1, 2, 3, 4) – values are allowed
globalPromptString	null	Text displayed in the prompt
waitFormName	WAITFRM	Wait form displayed after capturing
terminalId	null	Should be valid terminal serial number (12 digit number). If property value is not specified, then terminal id is taken from current IODriverOptions. Taken into account only for communication via Ethernet port
bufferSize	null	(optional) Integer property, represents size of signature buffer in bytes (FPE interface Form Request 'V' "SB" token)
uuEncodedFlag	false	(optional) Boolean flag that identifies whether to use UUEncoding for Signature capture (FPE interface Form Request 'V' "SU" token)
messageEnabledFlag	true	(optional) Boolean flag that identifies whether to enable or disable signature capture message (FPE interface Form Request 'V' "SS" token)

Methods

Method	Description
beginCapture (formName: *string*): void { raises-exception, use after open-claim-enable }	Executes FPETerminalInterface.beginSignatureCapture with parameters specified in static parameters
endCapture ( ): void { raises-exception, use after open-claim-enable }	Terminates signature capture. Shows Wait Form with text “Capture terminated”

Properties

Property	Value	Description
CapDisplay: *boolean* { read-only } 1.0 open	True	
CapRealTimeData: *boolean* { read-only } 1.2 open	False	
CapUserTerminated: *boolean* { read-only } 1.0 open	True	User can terminate signature capture by pressing Ok or Cancel button
MaximumX: *int32* { read-only } 1.0 open	Returns X Axis Resolution static property value	
MaximumY: *int32* { read-only } 1.0 open	Returns Y Axis Resolution static property value	
PointArray: *array of points* { read-only } 1.0 open, claim, & enable		As in Unified POS specification
RawData: *binary* { read-only } 1.0 open, claim, & enable		As in Unified POS specification
RealTimeDataEnabled: *boolean* { read-write } 1.2 open	False	Throws E_ILLEGAL Cannot set to true because **CapRealTimeData** is false.

LOGGING SUPPORT

The FPEInterface and JPOS services perform logging during working process. To switch logging on it's necessary to set log level to value that differs from "LOG_NO" (See table below). To set log level it's necessary to set **-Dcom.hypercom.util.LogLevel=<lenel_no>** option of java machine command line.

Log Level	Value	Notes
LOG_NO	0	***No log is written***
LOG_ERROR	1	Only error messages are written
LOG_INFO	2	Informational and error messages are written
LOG_DEBUG	3	All messages are written including detailed debug messages

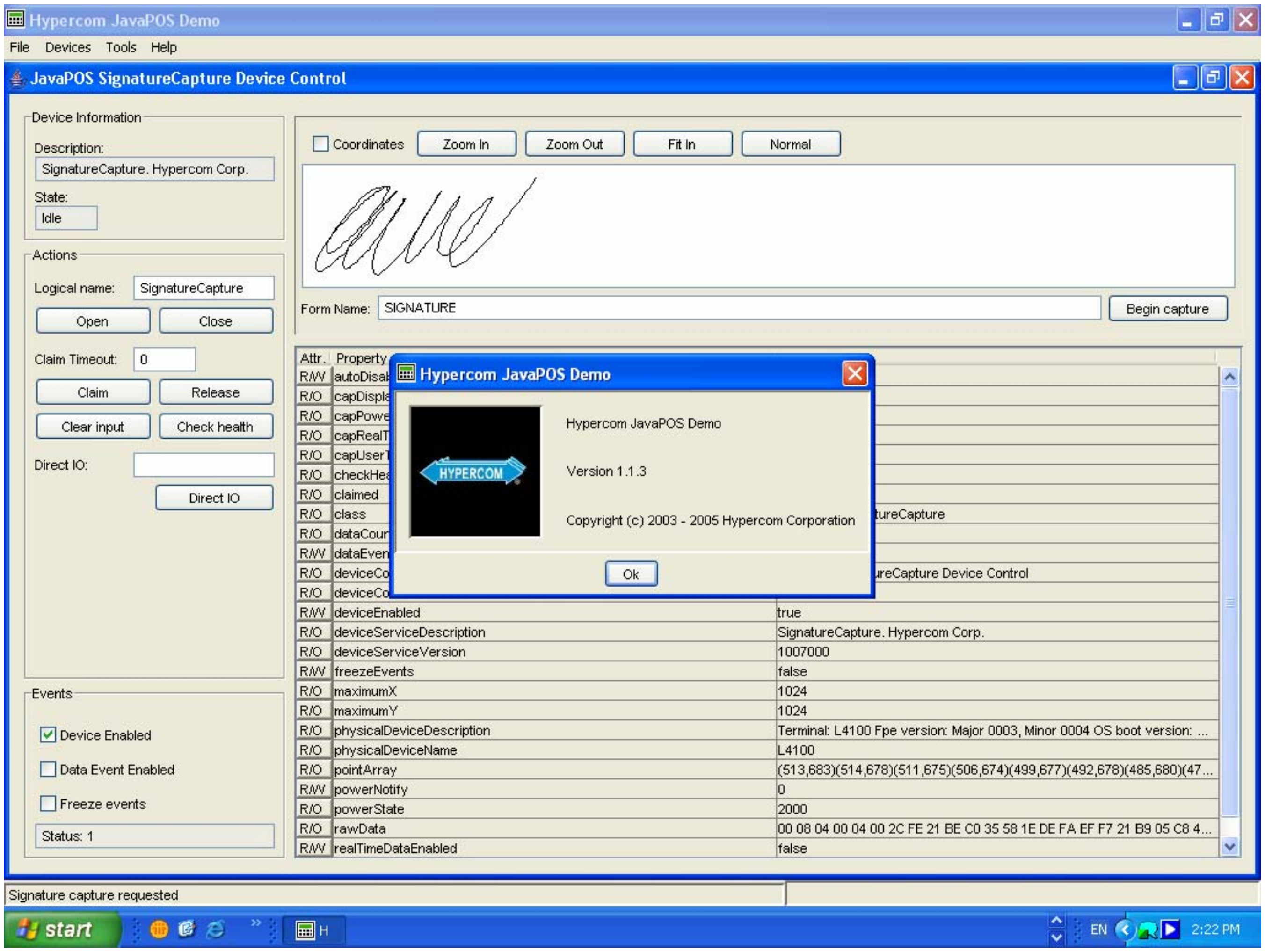
USE OF JPOS SERVICES IN JAVA-BASED POS APPLICATIONS

To use JPOS service it is necessary to initialize JCL registry first. It is initialized from XML file containing definitions of service classes to be used. The sample code below shows one of the ways how to do it. This sample uses JCL.xml file. The file configured for Symbol services is distributed together with JPOS service implementation library. See Appendix D for MSR control usage sample code.

```
 import jpos.loader.*;
 import jpos.*;


public static void main(String[] args) {
    // Initializing JCL registry
    try {
      System.setProperty("jpos.config.populatorFile"/**/, "JCL.xml");
      JposServiceManager manager = JposServiceLoader.getManager();
      manager.getEntryRegistry().load();
    } catch (Exception ex) {
      ex.printStackTrace();
      return;
    }

    // Using JPOS controls
   try {
        PINPad pinPad = new PINPad();
        PinPad.open(“PINPad”);  // The logical name “PINPad”
                                            // must have correspondent entry in JCL.xml file

        // Call other functions of pinpad ….
        // See UnifiedPOS specification for more details
        // About them

    } catch(Exception ex) {
        ex.printStackTrace();
    }
}
```

JPOS SERVICE TEST APPLICATION

Symbol also provides a JPOS service test application demonstrating how the implemented services work.



Device Information

Description: SignatureCapture. Hypercom Corp.

State: Idle

Actions

Logical name: SignatureCapture

Claim Timeout: 0

Direct IO:

Events

[✓] Device Enabled

[ ] Data Event Enabled

[ ] Freeze events

Status: 1

Form Name: SIGNATURE

Attr.	Property	
R/W	autoDisal	
R/O	capDispla	
R/O	capPowe	
R/O	capRealT	
R/O	capUserT	
R/O	checkHea	
R/O	claimed	
R/O	class	tureCapture
R/O	dataCour	
R/W	dataEven	
R/O	deviceCo	ureCapture Device Control
R/O	deviceCo	
R/W	deviceEnabled	true
R/O	deviceServiceDescription	SignatureCapture. Hypercom Corp.
R/O	deviceServiceVersion	1007000
R/W	freezeEvents	false
R/O	maximumX	1024
R/O	maximumY	1024
R/O	physicalDeviceDescription	Terminal: L4100 Fpe version: Major 0003, Minor 0004 OS boot version: ...
R/O	physicalDeviceName	L4100
R/O	pointArray	(513,683)(514,678)(511,675)(506,674)(499,677)(492,678)(485,680)(47...
R/W	powerNotify	0
R/O	powerState	2000
R/O	rawData	00 08 04 00 04 00 2C FE 21 BE C0 35 58 1E DE FA EF F7 21 B9 05 C8 4...
R/W	realTimeDataEnabled	false

Hypercom JavaPOS Demo

Hypercom JavaPOS Demo

Version 1.1.3

Copyright (c) 2003 - 2005 Hypercom Corporation

Signature capture requested

Figure 6 *JavaPOS service demo provided by Symbol.*

Appendix A Sample Applications

There are 6 applications coming together with JPOS service implementation.

1. **Start IODriver** (`JCommApp.exe` – on Windows, `JCommApp` – on Linux) - Executes IODriver application that is central connection point to POS terminal for other applications.
2. **Comm Demo** (`JCommDemo.exe` – on Windows, `JCommDemo` – on Linux)
   This is GUI demo working directly with IODriver. Allows sending messages to IODriver and monitors incoming messages. Also allows changing of IODriver parameters. If this application is run after IODriver is started then communication is performed through IODriver initialized in JCommApp.exe, otherwise it starts its own in-process IODriver.
3. **JPos Demo** (`JPosDemo.exe` – on Windows, `JPosDemo` – on Linux)
   Executes JavaPOS demo application demonstrating work of JavaPOS object implementations.
   If this application is ran after IODriver is started (e.g. JCommApp.exe - on Windows platform, JCommApp - on Linux) then communication is performed through IODriver initialized in JCommApp, otherwise it starts its own in-process IODriver.
4. **Java FPE-Sim** (`JFPEDemo.exe` – on Windows, `JavaFPE-Sim` – on Linux)
   Executes FPEInterface demonstration application.
   If this application is run after IODriver is started then communication is performed through IODriver initialized in JCommApp, otherwise it starts its own in-process IODriver.
5. **Trace** (`tracer.exe` – on Windows, `tracer` – on Linux)
   Executes JavaPOS Tracing application showing raw protocol messages sent to/from POS terminal in hexadecimal format. Can be used to monitor all messages send/received from the connected terminals
6. **JCL Editor** (`jcleditor.exe` – on Windows, `jcleditor` – on Linux)
   Executes JCL file editor coming together with JCL library. (This is not a Symbol product.)

To start JavaPOS applications on Windows, do one of the following:

Run `<application_name>.exe` (e.g. `JPosDemo.exe`) from bin directory under the one where you installed the JavaPOS application (by default it's `C:\Program Files\ HypercomJavaPOS\ bin`).

Choose **Start|Programs|Hypercom| `<application_name>`** (e.g. **JPos Demo**)

To start JavaPOS applications installed on Linux, do one of the following:

Run `<application_name>` script from any directory of Shell or Root Console (Konsole). Works fine for JavaPOS applications installed as RPM package and by Unix/Linux GUI installer (provided Symlinks were successfully added to default directory suggested by installation, otherwise call from `bin` directory)

e.g.: [root@hypercom HypercomJavaPOS]# `JPosDemo`

Double-click on application shell script (`<application_name> - e.g.` **JPosDemo**) from `bin` directory under the one where you installed the JavaPOS application.

Double-click on `<application_name>.desktop` (e.g. `JPosDemo.desktop`) Desktop Config File from the directory where you installed the JavaPOS application (you may link this files to your user Desktop if needed).

✓ **Note:** The application launchers execute Java machine referring to class libraries residing in the "lib" directory. If it is necessary to run programs with some different parameters or integrate into other applications, the libraries should be added in the class path.

Appendix B Sample JCL.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE JposEntries PUBLIC "-//JavaPOS//DTD//EN"
                             "jpos/res/jcl.dtd">
<JposEntries>
<!--Saved by JavaPOS jpos.config/loader (JCL) version 2.1.0-RC3 on 5/28/05 3:11
PM-->

    <JposEntry logicalName="Keyboard">
        <creation factoryClass="com.hypercom.fpe.jpos.ServiceInstanceFactory"
serviceClass="com.hypercom.fpe.jpos.keyboard.KeyboardService"/>
        <vendor name="Hypercom Corporation" url="http://www.hypercom.com"/>
        <jpos category="LineDisplay" version="1.7"/>
        <product description="Hypercom JavaPOS for FPE terminal" name="Hypercom
JavaPOS for FPE terminal" url="http://www.hypercom.com"/>

        <!--Other non JavaPOS required property (mostly vendor properties and
bus specific properties i.e. RS232 )-->
        <prop name="deviceBus" type="String" value="Unknown"/>
</JposEntry>


    <JposEntry logicalName="NewSignatureCapture">
        <creation factoryClass="com.hypercom.fpe.jpos.ServiceInstanceFactory"
serviceClass="com.hypercom.fpe.jpos.sigcap.SignatureCaptureService"/>
        <vendor name="Hypercom Corporation" url="http://www.hypercom.com"/>
        <jpos category="LineDisplay" version="1.7"/>
        <product description="Hypercom JavaPOS for FPE terminal" name="Hypercom
JavaPOS for FPE terminal" url="http://www.hypercom.com"/>

        <!--Other non JavaPOS required property (mostly vendor properties and
bus specific properties i.e. RS232 )-->
        <prop name="terminalId" type="String" value="100004541886"/>
        <prop name="globalPromptString" type="String" value="Please Sign
Below"/>
        <prop name="globalPromptNumber" type="Integer" value="1"/>
        <prop name="bufferSize" type="Integer" value="99999"/>
        <prop name="penUpTimeout" type="Integer" value="9"/>
        <prop name="uuEncodedFlag" type="Boolean" value="true"/>
        <prop name="waitFormName" type="String" value="WAITFRM"/>
        <prop name="resolution" type="String" value="High"/>
        <prop name="messageEnabledFlag" type="Boolean" value="true"/>
        <prop name="deviceBus" type="String" value="Unknown"/>
</JposEntry>


    <JposEntry logicalName="PINPad">
        <creation factoryClass="com.hypercom.fpe.jpos.ServiceInstanceFactory"
serviceClass="com.hypercom.fpe.jpos.pinpad.PINPadService"/>
        <vendor name="Hypercom Corporation" url="http://www.hypercom.com"/>
        <jpos category="LineDisplay" version="1.7"/>
        <product description="Hypercom JavaPOS for FPE terminal" name="Hypercom
JavaPOS for FPE terminal" url="http://www.hypercom.com"/>

        <!--Other non JavaPOS required property (mostly vendor properties and
bus specific properties i.e. RS232 )-->
        <prop name="PPAD_MSG_SELECTCARDTYPE_G2" type="String"
value="PPAD_MSG_SELECTCARDTYPE_G2"/>
        <prop name="PPAD_MSG_SELECTCARDTYPE_G1" type="String"
value="PPAD_MSG_SELECTCARDTYPE_G1"/>
        <prop name="PPAD_MSG_PLEASEWAIT_FORM" type="String" value="WAITFRM"/>
        <prop name="PPAD_MSG_ENTERPIN_FORM" type="String" value="PIN"/>
        <prop name="PPAD_MSG_IDLE_FORM" type="String" value="IDLEFRM"/>
        <prop name="PPAD_MSG_INSERTCARD_FORM" type="String" value="INSERTFRM"/>
        <prop name="PPAD_MSG_CANCELED_G2" type="String" 
value="PPAD_MSG_CANCELED_G2"/>
        <prop name="PPAD_MSG_CANCELED_G1" type="String" 
value="PPAD_MSG_CANCELED_G1"/>
        <prop name="globalPromptNumber" type="Integer" value="1"/>
        <prop name="PPAD_MSG_APPROVED_FORM" type="String" value="APPROVFRM"/>
        <prop name="PPAD_MSG_RETRIESEXCEEDED_FORM" type="String" 
value="RETEXCFRM"/>
        <prop name="deviceBus" type="String" value="HID"/>
        <prop name="PPAD_MSG_DECLINED_FORM" type="String" value="DECLFRM"/>
        <prop name="PPAD_MSG_SLIDE_CARD_G2" type="String" 
value="PPAD_MSG_SLIDE_CARD_G2"/>
        <prop name="PPAD_MSG_SLIDE_CARD_G1" type="String" 
value="PPAD_MSG_SLIDE_CARD_G1"/>
        <prop name="globalPromptString" type="String" value="Please Enter 
PIN"/>
        <prop name="PPAD_MSG_DECLINED_G2" type="String" 
value="PPAD_MSG_DECLINED_G2"/>
        <prop name="pinFormName" type="String" value="PINFRM"/>
        <prop name="PPAD_MSG_DECLINED_G1" type="String" 
value="PPAD_MSG_DECLINED_G1"/>
        <prop name="cashbackEnabled" type="Boolean" value="true"/>
        <prop name="PPAD_MSG_ENTERVALIDPIN_G2" type="String" 
value="PPAD_MSG_ENTERVALIDPIN_G2"/>
        <prop name="PPAD_MSG_SELECTCARDTYPE_FORM" type="String" 
value="GETTENDFRM"/>
        <prop name="PPAD_MSG_ENTERVALIDPIN_G1" type="String" 
value="PPAD_MSG_ENTERVALIDPIN_G1"/>
        <prop name="PPAD_MSG_NOTREADY_FORM" type="String" value="NOTREADY"/>
        <prop name="PPAD_MSG_ENTERPIN_G2" type="String" 
value="PPAD_MSG_ENTERPIN_G2"/>
        <prop name="PPAD_MSG_ENTERPIN_G1" type="String" 
value="PPAD_MSG_ENTERPIN_G1"/>
        <prop name="promptFormName" type="String" value="SWIPEFRM"/>
        <prop name="masterKeyIndex" type="Integer" value="1"/>
        <prop name="PPAD_MSG_CANCELED_FORM" type="String" value="CANCELFRM"/>
        <prop name="PPAD_MSG_RETRIESEXCEEDED_G2" type="String" 
value="PPAD_MSG_RETRIESEXCEEDED_G2"/>
        <prop name="workingKey" type="String" value="1234567123454789"/>
        <prop name="PPAD_MSG_RETRIESEXCEEDED_G1" type="String" 
value="PPAD_MSG_RETRIESEXCEEDED_G1"/>
        <prop name="PPAD_MSG_APPROVED_G2" type="String" 
value="PPAD_MSG_APPROVED_G2"/>
        <prop name="PPAD_MSG_APPROVED_G1" type="String" 
value="PPAD_MSG_APPROVED_G1"/>
        <prop name="PPAD_MSG_ENTERPIN" type="Long" value="1"/>
        <prop name="PPAD_MSG_IDLE_G2" type="String" value="PPAD_MSG_IDLE_G2"/>
        <prop name="PPAD_MSG_IDLE_G1" type="String" value="PPAD_MSG_IDLE_G1"/>
        <prop name="PPAD_MSG_INSERTCARD_G2" type="String" 
value="PPAD_MSG_INSERTCARD_G2"/>
        <prop name="PPAD_MSG_INSERTCARD_G1" type="String" 
value="PPAD_MSG_INSERTCARD_G1"/>
        <prop name="PPAD_MSG_NOTREADY_G2" type="String" 
value="PPAD_MSG_NOTREADY_G2"/>
        <prop name="PPAD_MSG_NOTREADY_G1" type="String" 
value="PPAD_MSG_NOTREADY_G1"/>
        <prop name="PPAD_MSG_SLIDE_CARD_FORM" type="String" value="SWIPEFRM"/>
        <prop name="PPAD_MSG_AMOUNTOK_G2" type="String" 
value="PPAD_MSG_AMOUNTOK_G2"/>
```

```
        <prop name="PPAD_MSG_AMOUNTOK_G1" type="String"
value="PPAD_MSG_AMOUNTOK_G1"/>
        <prop name="PPAD_MSG_AMOUNTOK_FORM" type="String" value="CBYNFRM"/>
        <prop name="PPAD_MSG_ENTERVALIDPIN_FORM" type="String"
value="VALPINFRM"/>
        <prop name="PPAD_MSG_PLEASEWAIT_G2" type="String"
value="PPAD_MSG_PLEASEWAIT_G2"/>
        <prop name="PPAD_MSG_PLEASEWAIT_G1" type="String"
value="PPAD_MSG_PLEASEWAIT_G1"/>
</JposEntry>


    <JposEntry logicalName="NewPINPad">
        <creation factoryClass="com.hypercom.fpe.jpos.ServiceInstanceFactory"
serviceClass="com.hypercom.fpe.jpos.pinpad.PINPadService"/>
        <vendor name="Hypercom Corporation" url="http://www.hypercom.com"/>
        <jpos category="LineDisplay" version="1.7"/>
        <product description="Hypercom JavaPOS for FPE terminal" name="Hypercom
JavaPOS for FPE terminal" url="http://www.hypercom.com"/>

        <!--Other non JavaPOS required property (mostly vendor properties and
bus specific properties i.e. RS232 )-->
        <prop name="PPAD_MSG_SELECTCARDTYPE_G2" type="String"
value="PPAD_MSG_SELECTCARDTYPE_G2"/>
        <prop name="PPAD_MSG_SELECTCARDTYPE_G1" type="String"
value="PPAD_MSG_SELECTCARDTYPE_G1"/>
        <prop name="PPAD_MSG_PLEASEWAIT_FORM" type="String" value="WAITFRM"/>
        <prop name="PPAD_MSG_ENTERPIN_FORM" type="String" value="PIN"/>
        <prop name="PPAD_MSG_IDLE_FORM" type="String" value="IDLEFRM"/>
        <prop name="PPAD_MSG_INSERTCARD_FORM" type="String" value="INSERTFRM"/>
        <prop name="PPAD_MSG_CANCELED_G2" type="String"
value="PPAD_MSG_CANCELED_G2"/>
        <prop name="PPAD_MSG_CANCELED_G1" type="String"
value="PPAD_MSG_CANCELED_G1"/>
        <prop name="globalPromptNumber" type="Integer" value="1"/>
        <prop name="PPAD_MSG_APPROVED_FORM" type="String" value="APPROVFRM"/>
        <prop name="PPAD_MSG_RETRIESEXCEEDED_FORM" type="String"
value="RETEXCFRM"/>
        <prop name="deviceBus" type="String" value="HID"/>
        <prop name="PPAD_MSG_DECLINED_FORM" type="String" value="DECLFRM"/>
        <prop name="PPAD_MSG_SLIDE_CARD_G2" type="String"
value="PPAD_MSG_SLIDE_CARD_G2"/>
        <prop name="PPAD_MSG_SLIDE_CARD_G1" type="String"
value="PPAD_MSG_SLIDE_CARD_G1"/>
        <prop name="globalPromptString" type="String" value="Please Enter
PIN"/>
        <prop name="PPAD_MSG_DECLINED_G2" type="String"
value="PPAD_MSG_DECLINED_G2"/>
        <prop name="pinFormName" type="String" value="PINFRM"/>
        <prop name="PPAD_MSG_DECLINED_G1" type="String"
value="PPAD_MSG_DECLINED_G1"/>
        <prop name="cashbackEnabled" type="Boolean" value="true"/>
        <prop name="PPAD_MSG_ENTERVALIDPIN_G2" type="String"
value="PPAD_MSG_ENTERVALIDPIN_G2"/>
        <prop name="PPAD_MSG_SELECTCARDTYPE_FORM" type="String"
value="GETTENDFRM"/>
        <prop name="PPAD_MSG_ENTERVALIDPIN_G1" type="String"
value="PPAD_MSG_ENTERVALIDPIN_G1"/>
        <prop name="PPAD_MSG_NOTREADY_FORM" type="String" value="NOTREADY"/>
        <prop name="PPAD_MSG_ENTERPIN_G2" type="String"
value="PPAD_MSG_ENTERPIN_G2"/>
```

```
        <prop name="PPAD_MSG_ENTERPIN_G1" type="String"
value="PPAD_MSG_ENTERPIN_G1"/>
        <prop name="masterKeyIndex" type="Integer" value="1"/>
        <prop name="promptFormName" type="String" value="SWIPEFRM"/>
        <prop name="PPAD_MSG_CANCELED_FORM" type="String" value="CANCELFRM"/>
        <prop name="PPAD_MSG_RETRIESEXCEEDED_G2" type="String"
value="PPAD_MSG_RETRIESEXCEEDED_G2"/>
        <prop name="PPAD_MSG_APPROVED_G2" type="String"
value="PPAD_MSG_APPROVED_G2"/>
        <prop name="PPAD_MSG_RETRIESEXCEEDED_G1" type="String"
value="PPAD_MSG_RETRIESEXCEEDED_G1"/>
        <prop name="workingKey" type="String" value="1234567123454789"/>
        <prop name="PPAD_MSG_APPROVED_G1" type="String"
value="PPAD_MSG_APPROVED_G1"/>
        <prop name="PPAD_MSG_ENTERPIN" type="Long" value="1"/>
        <prop name="PPAD_MSG_IDLE_G2" type="String" value="PPAD_MSG_IDLE_G2"/>
        <prop name="PPAD_MSG_IDLE_G1" type="String" value="PPAD_MSG_IDLE_G1"/>
        <prop name="PPAD_MSG_INSERTCARD_G2" type="String"
value="PPAD_MSG_INSERTCARD_G2"/>
        <prop name="PPAD_MSG_INSERTCARD_G1" type="String"
value="PPAD_MSG_INSERTCARD_G1"/>
        <prop name="PPAD_MSG_NOTREADY_G2" type="String"
value="PPAD_MSG_NOTREADY_G2"/>
        <prop name="PPAD_MSG_NOTREADY_G1" type="String"
value="PPAD_MSG_NOTREADY_G1"/>
        <prop name="PPAD_MSG_SLIDE_CARD_FORM" type="String" value="SWIPEFRM"/>
        <prop name="terminalId" type="String" value="000000000000"/>
        <prop name="PPAD_MSG_AMOUNTOK_G2" type="String"
value="PPAD_MSG_AMOUNTOK_G2"/>
        <prop name="PPAD_MSG_AMOUNTOK_G1" type="String"
value="PPAD_MSG_AMOUNTOK_G1"/>
        <prop name="PPAD_MSG_AMOUNTOK_FORM" type="String" value="CBYNFRM"/>
        <prop name="PPAD_MSG_ENTERVALIDPIN_FORM" type="String"
value="VALPINFRM"/>
        <prop name="PPAD_MSG_PLEASEWAIT_G2" type="String"
value="PPAD_MSG_PLEASEWAIT_G2"/>
        <prop name="PPAD_MSG_PLEASEWAIT_G1" type="String"
value="PPAD_MSG_PLEASEWAIT_G1"/>
</JposEntry>


    <JposEntry logicalName="SignatureCapture">
        <creation factoryClass="com.hypercom.fpe.jpos.ServiceInstanceFactory"
serviceClass="com.hypercom.fpe.jpos.sigcap.SignatureCaptureService"/>
        <vendor name="Hypercom Corporation" url="http://www.hypercom.com"/>
        <jpos category="LineDisplay" version="1.7"/>
        <product description="Hypercom JavaPOS for FPE terminal" name="Hypercom
JavaPOS for FPE terminal" url="http://www.hypercom.com"/>

        <!--Other non JavaPOS required property (mostly vendor properties and
bus specific properties i.e. RS232 )-->
        <prop name="deviceBus" type="String" value="Unknown"/>
        <prop name="resolution" type="String" value="High"/>
        <prop name="globalPromptNumber" type="Integer" value="1"/>
        <prop name="globalPromptString" type="String" value="Please Sign
Below"/>
        <prop name="waitFormName" type="String" value="WAITFRM"/>
        <prop name="penUpTimeout" type="Integer" value="9"/>
</JposEntry>


    <JposEntry logicalName="MSR">
```

```
        <creation factoryClass="com.hypercom.fpe.jpos.ServiceInstanceFactory"
serviceClass="com.hypercom.fpe.jpos.msr.MSRService"/>
        <vendor name="Hypercom Corporation" url="http://www.hypercom.com"/>
        <jpos category="MSR" version="1.7"/>
        <product description="Hypercom JavaPOS for FPE terminal" name="Hypercom
JavaPOS for FPE terminal" url="http://www.hypercom.com"/>

        <!--Other non JavaPOS required property (mostly vendor properties and
bus specific properties i.e. RS232 )-->
        <prop name="swipeFormName" type="String" value="SWIPEFRM"/>
        <prop name="deviceBus" type="String" value="Unknown"/>
        <prop name="gloablPromptString2" type="String" value="OR USE
SPEEDPASS"/>
        <prop name="gloablPromptString1" type="String" value="PLEASE SLIDE YOUR
CARD"/>
</JposEntry>


    <JposEntry logicalName="NewLineDisplay">
        <creation factoryClass="com.hypercom.fpe.jpos.ServiceInstanceFactory"
serviceClass="com.hypercom.fpe.jpos.linedisplay.LineDisplayService"/>
        <vendor name="Hypercom Corporation" url="http://www.hypercom.com"/>
        <jpos category="LineDisplay" version="1.7"/>
        <product description="Hypercom JavaPOS for FPE terminal" name="Hypercom
JavaPOS for FPE terminal" url="http://www.hypercom.com"/>

        <!--Other non JavaPOS required property (mostly vendor properties and
bus specific properties i.e. RS232 )-->
        <prop name="terminalId" type="String" value="000000000000"/>
        <prop name="deviceBus" type="String" value="Unknown"/>
        <prop name="foregroundColor" type="Integer" value="0"/>
        <prop name="backgroundColor" type="Integer" value="255"/>
        <prop name="fontSize" type="Integer" value="0"/>
</JposEntry>


    <JposEntry logicalName="NewMSR">
        <creation factoryClass="com.hypercom.fpe.jpos.ServiceInstanceFactory"
serviceClass="com.hypercom.fpe.jpos.msr.MSRService"/>
        <vendor name="Hypercom Corporation" url="http://www.hypercom.com"/>
        <jpos category="MSR" version="1.7"/>
        <product description="Hypercom JavaPOS for FPE terminal" name="Hypercom
JavaPOS for FPE terminal" url="http://www.hypercom.com"/>

        <!--Other non JavaPOS required property (mostly vendor properties and
bus specific properties i.e. RS232 )-->
        <prop name="swipeFormName" type="String" value="SWIPEFRM"/>
        <prop name="terminalId" type="String" value="000000000000"/>
        <prop name="deviceBus" type="String" value="Unknown"/>
        <prop name="gloablPromptString2" type="String" value="OR USE
SPEEDPASS"/>
        <prop name="gloablPromptString1" type="String" value="PLEASE SLIDE YOUR
CARD"/>
</JposEntry>


    <JposEntry logicalName="NewKeyboard">
        <creation factoryClass="com.hypercom.fpe.jpos.ServiceInstanceFactory"
serviceClass="com.hypercom.fpe.jpos.keyboard.KeyboardService"/>
        <vendor name="Hypercom Corporation" url="http://www.hypercom.com"/>
        <jpos category="LineDisplay" version="1.7"/>
        <product description="Hypercom JavaPOS for FPE terminal" name="Hypercom
JavaPOS for FPE terminal" url="http://www.hypercom.com"/>

        <!--Other non JavaPOS required property (mostly vendor properties and
bus specific properties i.e. RS232 )-->
        <prop name="terminalId" type="String" value="000000000000"/>
        <prop name="deviceBus" type="String" value="Unknown"/>
</JposEntry>


    <JposEntry logicalName="LineDisplay">
        <creation factoryClass="com.hypercom.fpe.jpos.ServiceInstanceFactory"
serviceClass="com.hypercom.fpe.jpos.linedisplay.LineDisplayService"/>
        <vendor name="Hypercom Corporation" url="http://www.hypercom.com"/>
        <jpos category="LineDisplay" version="1.7"/>
        <product description="Hypercom JavaPOS for FPE terminal" name="Hypercom
JavaPOS for FPE terminal" url="http://www.hypercom.com"/>

        <!--Other non JavaPOS required property (mostly vendor properties and
bus specific properties i.e. RS232 )-->
        <prop name="deviceBus" type="String" value="Unknown"/>
        <prop name="foregroundColor" type="Integer" value="0"/>
        <prop name="backgroundColor" type="Integer" value="255"/>
        <prop name="fontSize" type="Integer" value="0"/>
</JposEntry>

</JposEntries>
```

Appendix C CommSettings.xml - Sample IODriver options file

```
<?xml version="1.0"?>
<comm-app-options comm-type="1" registry-port="4209">
    <IODriverUrl>//localhost:4209/iodriver</IODriverUrl>
    <tcp-iPOptions TCPIPPort="5110" comm-type="1">
        <TCPIPhost>localhost</TCPIPhost>
    </tcp-iPOptions>
    <active-iODriver-options TCPIPPort="5110" comm-type="1"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="java:com.hypercom.commservice.tcpip.TCPIPOptions">
        <TCPIPhost>localhost</TCPIPhost>
    </active-iODriver-options>
    <log-options log-level="3" log-output="0">
        <log-file></log-file>
    </log-options>
    <serial-options comm-type="0" stop-bits="1" data-bits="8" parity="0" baud-
rate="19200">
        <serial-port>COM1</serial-port>
    </serial-options>
</comm-app-options>
```

Appendix D Sample MSR control usage

```
import jpos.*;
import jpos.loader.JposServiceManager;
import jpos.loader.JposServiceLoader;


/**
 * <p><b>Title:</b> Hypercom JavaPOS library</p>
 * <p><b>Description:</b>
 * This sample demonstrates the usage of
 * JPOS MSR device.  To run this sample
 * its necessary to execute "jcommapp" batch file
 * that starts IODriver application.
 * </p>
 * <p><b>Copyright:</b> Copyright (c) 2003 - 2006</p>
 * <p><b>Company:</b> Hypercom</p>
 * @author Jelena Lubimova
 * @version $Revision:  $
 */
public abstract class SampleMSRUsage {
  public static void main(String[] args) {
    // Initializing JCL registry
    try {
      System.setProperty("jpos.config.populatorFile" /**/, "JCL.xml");
      JposServiceManager manager = JposServiceLoader.getManager();
      manager.getEntryRegistry().load();
    }
    catch (Exception ex) {
      ex.printStackTrace();
      return;
    }

    // Using LineDisplay
    MSR msr = new MSR();
    try {

      // Initializing device
      msr.open("MSR");

      msr.claim(5000);
      msr.setDeviceEnabled(true);
      msr.setDataEventEnabled(true);

      Object obj = new Object();

      String strDirectIO = "TS";
      int[] buff = new int[strDirectIO.length()];
      for (int i = 0; i < strDirectIO.length(); ++i) {
       //filling buffer with string, which later will be passed as "directIO"
       //method "data" parameter
       buff[i] = strDirectIO.charAt(i);
      }
      try {
       // Waiting until message received
       Thread.sleep(5000);
      }
      catch (Exception e) {
       e.printStackTrace();
      }

      /**
```

```
       * directIO Method Syntax:
       * directIO ( command: int32, inout data: int32, inout obj: object ):void
{ raises-exception }
       * Comments:  When directIO method called on MSR control is passed "data"
parameter equal to "TS",
       * then the first element of "data" (data[0]) will contain
TrackDataSource code as
       * it is described in FPE Interface specification.
       * E.g. ‘0’ - TRACK_READER
       * ‘1’ - RDIF_EXPRESS_PAY
       * ‘2’ - RFID_PAY_PASS
       * ‘3’ - RFID_VISA_WAVE
       * ‘4’ - RFID_UNKNOWN_APP
       */
      //calling directIO method on MSR control with "TS" passed as "data"
parameter
      msr.directIO(1, buff, obj);

      //after directIO method execution first element of "data" array will
contain buff[0]
      System.out.println("Track Data source = " + buff[0]);

      // Finishing session
      msr.close();

    }
    catch (JposException ex) {
      ex.printStackTrace();
    }
  }
}
```

References

1 UnifiedPOS Retail Peripheral Architecture Version 1.7, July 24, 2002, http://www.nrf-arts.org/
2 Hypercom JavaPOS for FPE terminals Version 1.1: Hypercom, February, 2005.
3 FPE Interface Specification Version 3.21 April 21, 2006

Symbol Technologies, Inc.
One Symbol Plaza
Holtsville, New York 11742-1300
http://www.symbol.com

72E-91668-01
Revision A - November 2006